RL-TR-97-202, Volume I (of two)
Final Technical Report
October 1997

MOLECULAR DYNAMICS SIMULATION UPGRADE

Synectics Corporation

Lisa Kolek and Geraldine W. Rogers

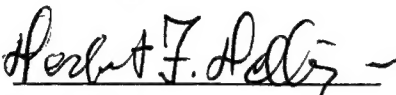*APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.*

19980414 158

DTIC QUALITY INSPECTED 4

Rome Laboratory
Air Force Materiel Command
Rome, New York

This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RL-TR-97-202, Volume I has been reviewed and is approved for publication.

APPROVED:

HERBERT F. HELBIG
Project Engineer

FOR THE DIRECTOR:

JOHN J. BART
Acting Director, Reliability
Electromagnetics & Reliability Directorate

If your address has changed or if you wish to be removed from the Rome Laboratory mailing list, or if the addressee is no longer employed by your organization, please notify RL/ERDR, 525 Brooks Rd, Rome, NY 13441-4505. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	October 1997	Final Jan 97 - Jun 97

4. TITLE AND SUBTITLE

MOLECULAR DYNAMICS SIMULATION UPGRADE

5. FUNDING NUMBERS

C - F30602-95-D-0028/0009
PE - 61102F
OR - 2982
TA - QE
WU - 10

6. AUTHOR(S)

Lisa Kolek and Geraldine W. Rogers

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Synectics Corporation
111 East Chestnut Street
Rome NY 13440-2831

8. PERFORMING ORGANIZATION REPORT NUMBER

WH-94-RY-09

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Rome Laboratory/ERDR
525 Brooks Rd
Rome NY 1344-4505

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

RL-TR-97-202, Volume I (of two)

11. SUPPLEMENTARY NOTES

Rome Laboratory Project Engineer: Herbert F. Helbig/ERDR/(315) 330-3495

12a. DISTRIBUTION AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The purpose of the MDEM simulation upgrade was to improve performance, enhance the "user-friendly" features and generally clean up the source code for the MDEM simulation software tool. Performance improvement was primarily focused on the attainment of MDEM processing speedup. This was achieved at a low level by examining the existing code structure, as well as at a high level by modifying the simulation algorithm. Enhancement of user-friendly features included the implementation of a prototype graphical user interface (GUI) and the creation of the user's guide for both the command-line and GUI versions of MDEM. Code cleanup involved deletion of unnecessary code and comments, and reorganization of variables. In addition to this, miscellaneous software engineering and development support was provided to Rome Laboratory on an "as required" basis.

14. SUBJECT TERMS

Molecular Dynamics, Computer Simulation, Electronic Materials

15. NUMBER OF PAGES

60

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

TABLE OF CONTENTS

List of Exhibits

List of Tables

SECTION ONE
TASK OVERVIEW

This Scientific and Final Technical Report (FINAL) was prepared by Synectics Corporation for submission to Rome Laboratory/ERST in accordance with Contract Data Requirements List (CDRL) item A003 of the "Molecular Dynamics Simulation Upgrade" task of contract F30602–94–D–0028/0009. The performance period for the task spanned four months and encompassed the investigation and/or implementation of many technical concepts pertinent to the upgrade.

The code comprising the simulation is known as MDEM, which is an acronym for the "Molecular Dynamics of Electronic Materials." The simulation was created by Rome Laboratory (Rome Lab) engineers to study the effects of atomic motion on the useful life of microcircuits. Information gleaned from MDEM simulations will be used to help improve the longevity of circuits in Department of Defense (DoD) airframe platforms which must remain functional for a 20–year period.

1.1 OBJECTIVES OF THE EFFORT

The scope of the MDEM simulation upgrade was very broad. The objectives of the effort encompassed the following areas.

- ❏ Improvement of performance

- ❏ Enhancement of "user–friendly" features

- ❏ General code cleanup

Performance improvement was primarily focused on the attainment of MDEM processing speedup. This was achieved at a low level by examining the existing code structure, as well as at a high level by modifying the simulation algorithm. Enhancement of user–friendly features included the implementation of a prototype graphical user interface (GUI) and the creation of a user's guide for both the command–line and GUI versions of MDEM. Code cleanup involved deletion of unnecessary code and comments and reorganization of variables. In addition to this, miscellaneous software engineering and development support was provided to Rome Lab on an "as required" basis.

1.2 PURPOSE OF THE REPORT

In order to meet the objectives of the upgrade, preliminary evaluation of the software by the contractor was essential.   A substantial familiarization period was required to become comfortable with the basic physics concepts underlying MDEM, as well as the code structure used to implement these concepts. Given the scope of the effort, individual tasks were assessed and prioritized by the contractor and presented to Rome Lab for approval. Preliminary cleanup of the code and verification of the simulation output was necessary before attempting major software modifications.   This early preparatory work established the foundation for proper implementation of upgrade enhancements.

The methods used to improve the MDEM simulation were basically derived from three sources.

- ❏ Rome Laboratory knowledge and guidance

- ❏ Contractor experience

- ❏ External information

Though the high–level objectives of the task were clear, discussions with Rome Lab offered further direction regarding Physics concepts, areas of emphasis, feasibility, and data validation. Contractor experience in the areas of software engineering and algorithm development provided insight required for software performance improvement.   External information, such as documentation and tools found in textbooks and on the Internet, was used to supplement the core knowledge base.

1.3 SUMMARY OF THE CONTENTS AND ORGANIZATION OF THE REPORT

This report is intended to present information concerning the research and development conducted by the contractor in support of the MDEM simulation upgrade.

- ❏ Section 2 gives an overview of the MDEM simulation design; this will furnish sufficient background for subsequent technical descriptions.

- ❏ Sections 3 — 6 describe the specific methods of simulation improvement, including code optimization, algorithm optimization, user interface implementation, and additional software enhancements.

- ❏ Section 7 presents specific benchmarking information gathered over the course of the effort.

- ❏ Lessons learned from the upgrade are documented in Section 8.

❑ Many worthwhile enhancements, which could not be implemented due to time constraints, have been relegated to Section 9 as recommendations.

❑ Appendix A contains a list of acronyms used in this document.

❑ The Software User's Manual is bound as Volume II of this Scientific and Final Technical Report (FINAL).

MDEM is the calculation–intensive workhorse at the heart of an end–to–end molecular dynamics simulation process. Exhibit 1 shows MDEM interaction with data synthesis and data reformatting/visualization components. Data synthesis involves the generation of MDEM–compatible atom files, via software or manual methods. Rome Lab engineers use a customized tool written in Visual Basic to generate MDEM input data. For output data visualization, Rome Lab primarily relies on *RasMol*, a powerful freeware package which has elaborate data display capabilities. An Rome Lab–created software tool is required to reformat MDEM output files for compatibility with *RasMol*.

The MDEM simulation software is comprised of five major processing components.

❑ Data input

❑ Data validation

❑ Interatomic force calculation

❑ Integration

❑ Data output

Exhibit 1 is a pictorial view of MDEM and its components. Data input and validation are performed once at the start of the simulation. Interatomic force calculation and integration are part of an iterative process which continues until the target number of integration steps is reached. Data output is performed at user–specified intervals during the integration and also at the end of the simulation. The following paragraphs present a brief description of each of MDEM's five major components.

2.1 DATA INPUT

In MDEM, data input is accomplished via software which reads a formatted atom file. This atom file contains header data and atom data. The header data are comprised of both status and control parameters. Status parameters provide information on the history of the data contained in the file. Items such as progenitor file name and total elapsed simulation time are considered status parameters. Control parameters, such as number of integration steps and thermostat data, direct the processing flow of the simulation.

Exhibit 1. MDEM Functionality in End–to–End Molecular Dynamics Processing
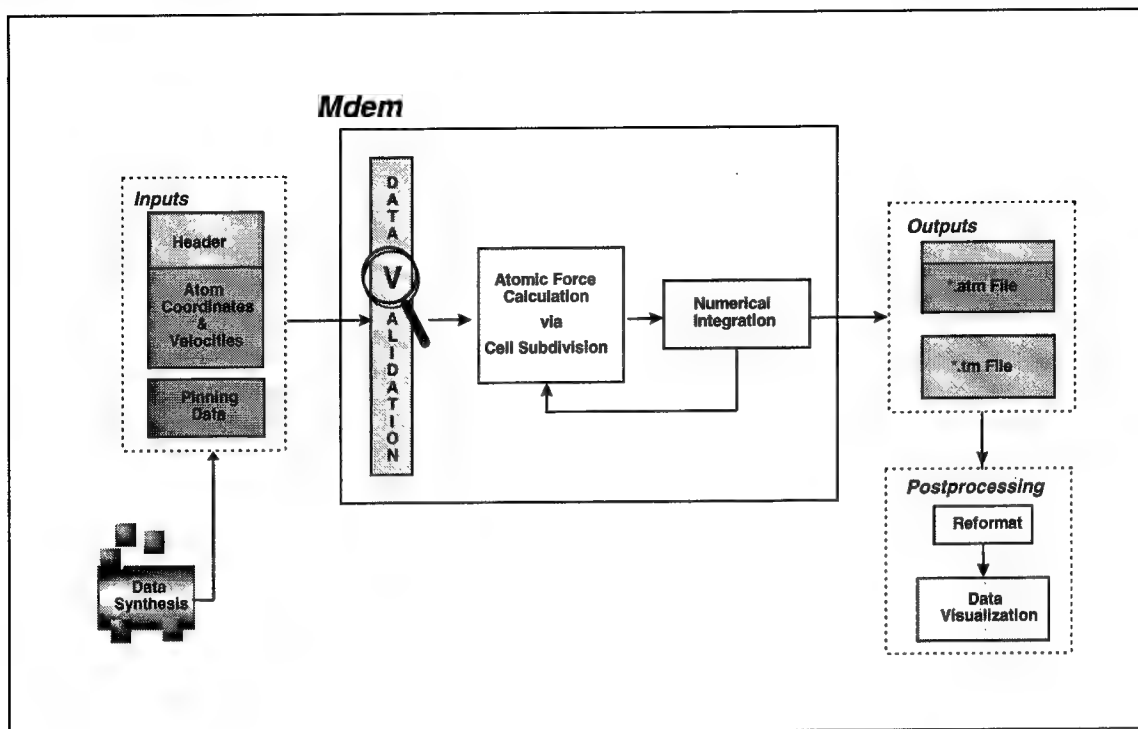
Following the header data are the atom data. For each atom, a line of information including position, velocity and atom tag is included. The position and velocity are specified in x, y, and z components. The atom tag is an encoded value which contains information on atomic number, pinning status, grain, and thermostat value. Information is extracted from the atom tag via appropriate masks and shifts. Throughout the atom file, predefined keywords are utilized to signal the presence of specific data. During the upgrade, an atom pinning capability was incorporated in the simulation, thus introducing a new type of input file containing pinning points (refer to Section 6.1.1 for additional information).

2.2 DATA VALIDATION

The validation component of MDEM examines individual data items for validity, as well as checking the dependencies among related data items. For example, the integration time step is declared invalid if it is set to zero. An example of data interdependency verification is illustrated with the pinning data. If an atom is designated as pinned, then the supporting data, such as pinning point and pinning force, are also checked for validity. In MDEM, fatal errors are flagged by the validation process and program execution is terminated.

2.3 INTERATOMIC FORCE CALCULATION

When an atom interacts with a neighboring atom, a force is exerted which is dependent upon the atom species and the distance between the atoms. The atomic motion resulting from these interactions can produce a change in overall system energy, or temperature. MDEM, which processes a system of atoms, must calculate and accumulate all the interatomic forces between neighboring atoms. In the simulation upgrade software, the Lennard–Jones force model was used to calculate interatomic forces. Near the end of the upgrade effort, Rome Lab engineers added a Stillenger–Weber force calculation module for additional flexibility.

The interatomic force calculation is typically the most time–consuming component of a molecular dynamics simulation. For large systems, the key to efficient processing lies in the method used to pair neighboring atoms. In the MDEM simulation, atom pairing techniques were the subject of much investigation. The atom interaction scheme used in the baseline code was replaced by a cell–division technique presented in The Art of Molecular Dynamics Simulation by D.C. Rapaport (refer to Section 4.1.2.1 for additional details).

2.4 INTEGRATION

The integration process produces velocity and acceleration data from the atom positions obtained via interatomic force calculation. Atom velocities are used to calculate system kinetic energy. Atom accelerations are used to determine total system work. The method of numerical integration utilized in MDEM is a second–order Taylor approximation. The processing involved in the integration component is minimal compared to that of the force calculation component. The number of integration steps controls the main looping mechanism in MDEM.

2.5 DATA OUTPUT

There are three types of output resulting from execution of the MDEM simulation.

- ❏ *Console data*, which are comprised of status messages and summary information, are output to the screen console to inform the user of simulation progress.

- ❏ *Atom files* are created at the frequency specified by the user via the write interval keyword in the input atom file. An atom file is always written after the last integration step has completed, regardless of the number specified by the user. The atom output files mirror the format of the atom input files, and can therefore be used as input to MDEM for subsequent processing.

❑ *Temperature files* are also created during the simulation execution. After each integration step, the system temperature, derived from kinetic energy and number of atoms, is written to the temperature file. During the upgrade, temperature file generation was made optional.

2.6   THE INITIAL MDEM BASELINE

The MDEM code received from Rome Lab at the commencement of the task served as the software baseline upon which all subsequent versions were built. A total of four data sets were received from Rome Lab for the purpose of testing the MDEM software. Two large data sets, consisting of 9,758 atoms and 112,423 atoms, were initially used to validate new software revisions against the baseline. Two small data sets, consisting of 13 atoms and 2 atoms, were later provided to test the algorithm modifications under extreme conditions. Though large data sets are typically used in molecular dynamics simulations, the small data sets did reveal algorithm weaknesses as well as floating point data related problems.

For the MDEM simulation upgrade, optimizations resulting in speedup were favored over those reducing memory requirements. The speedup achieved by code enhancement is dependent on many factors, including coding conventions, processor architecture, compiler quality, and compiler optimization switches used. Typically, code must be written for a specific platform in order to maximize performance. For the purposes of this effort, the primary target platform for the code optimization was the Intel Pentium Pro processor.

3.1 METHODS EMPLOYED

The methods used to optimize the MDEM code ranged from basic, intuitive simplification techniques, to utilization of recently developed, high–technology tools. The contractor's knowledge of efficient coding practices was the starting point for code optimization. This was supplemented by the use of documentation aimed specifically at the target simulation platform. In general, optimized code is only as good as the compiler which converts it into machine–level code. Therefore, a substantial amount of time was devoted to the investigation of various compilers and their native optimization switches. Most compilers are available with an associated profiling capability, which helps the user pinpoint frequently–executed areas of code for optimization. Two profilers were briefly investigated during the upgrade. A state–of–the–art code analysis tool, Intel's VTune, was also utilized during the code optimization effort.

3.1.1 EFFICIENT CODING PRACTICES

A developer typically acquires knowledge of efficient coding practices either through formal training or "on–the–job" experience. However, this knowledge is not always applicable, given the vast array of processor architectures available today. Code that is efficient on one processor could be totally ineffective on another. Therefore, it is always best to research the target processor in order to implement the most efficient code. For the MDEM upgrade, both visual inspection and documentation research were used to assess and improve the quality of the code.

3.1.1.1 VISUAL INSPECTION

Initial inspection of the MDEM code was performed to reveal areas which could be improved by general simplification and/or consolidation. The following list of potential code improvements was used as a guideline when inspecting the code.

- ❏ Conversion of divides to multiplies

- ❏ Elimination of redundant calculations

- ❏ Elimination of unnecessary variables

- ❏ Simplification of mathematical expressions

- ❏ Placement of static code outside loops

- ❏ Replacement of structures with scalars

- ❏ Simplification of array indexing

- ❏ Assignment of frequently used variables to registers

- ❏ Conversion to function inlining

- ❏ Simplification of data storage types

The baseline code was found to be fairly well–written with respect to these guidelines. Although several modifications were made according to this list, the impact on processing time was minimal. The reduction in processing time was greater on the Pentium than on the Pentium Pro due to major differences in the processor architectures. Also, today's sophisticated compilers often implement these optimizations automatically. It soon became apparent that "traditional" code optimization was not sufficient to obtain a significant speedup.

3.1.1.2 DOCUMENTATION

The Intel web site was searched for technical memos regarding code optimization for the Pentium and Pentium Pro processors. It was discovered that most of these deal with implementation at the assembly language level. Due to the complexity of the assembly level code and the time constraints of the MDEM upgrade task, these optimizations were not considered feasible options. The documentation regarding assembly optimization was noted but not utilized.

A technical memo called, "Optimization Strategies for the Pentium Processor®,"[1] briefly addresses the restructuring of C code to maximize optimization by the compiler. This memo summarizes the unique characteristics of the Pentium processor architecture and provides an overview of C–level optimization techniques. The memo states that the four primary coding areas which have a significant impact on compiler effectiveness involve:

---

[1] "Optimization Strategies for the Pentium Processor®", Intel Corporation,
http://support.intel.com/oem_developer/microprocessors/pentium/7299.HTM

❑ Pointer usage

❑ Global variable usage

❑ Loop implementation

❑ Data type mixing

A short description of proper coding techniques is given for each of the four areas. It was determined that the MDEM baseline code was already following these guidelines.

Another source of information regarding code performance improvement was found in the Microsoft™ Visual C++ "on–line" help utility. A technical memo called "Tips for Improving Time–Critical Code"[2] describes code efficiency issues from the Microsoft implementation perspective. This memo focuses on such areas as cache hits/page faults, sorting/searching, shared libraries, and thread implementation. Though geared toward the Microsoft development environment, the information contained in this memo is also valuable for general implementations as well.

3.1.2 COMPILERS

Selecting an appropriate compiler is not a trivial task. The most highly optimized C source code must still be translated to machine level by a compiler. The performance of the machine level code is directly related to the quality of the compiler, in addition to the compilation optimization switches chosen by the user. The two primary C compilers used during the MDEM simulation upgrade were DJGPP and Microsoft Visual C++. In addition to these, the Watcom and Intel compilers were briefly investigated.

3.1.2.1 DJGPP

DJGPP is the C compiler used by Rome Lab engineers to generate MDEM executables. The contractor also used DJGPP to generate timing statistics and to verify compatibility of changes made to software via Microsoft Visual C++. DJGPP is a freeware compiler used for developing 32–bit C/C++ software in the MS–DOS environment. DJGPP was created in an attempt to port the Unix gcc compiler to MS–DOS, and in the process, C++ capability was added. Although it is reported to have widespread distribution in many countries and languages, its usage for official applications should be carefully evaluated. There is some technical support for DJGPP; however, a user must typically wade through extensive literature in an attempt to find solutions to problems. There is no "help line" available for immediate, thorough support, as is the case

---

[2] "Tips for Improving Time–Critical Code", Microsoft Visual C++ on–line help

with a large company such as Microsoft. DJGPP also lacks a visual development environment as well as interface building capabilities.

The DJGPP compiler flags were tested during the upgrade effort. The following flags were exercised in an attempt to generate fast code.

❑ Compile flag –O1 — reduces code size and execution time of the program.

❑ Compile flag –O2 — includes all optional optimization, except loop unrolling and function inlining.

❑ Compile flag –O3 — includes all –O2 optimizations as well as function inlining.

3.1.2.2 MICROSOFT VISUAL C++

The Microsoft Visual C++ compiler was used by the contractor as the primary software development tool. This compiler is part of a sophisticated visual support environment which eases the difficulties encountered in application development. Makefiles are readily generated and maintained; debug and release version folders are automatically created; and source code is highlighted in various colors to indicate functionality. Also, development capabilities exist for generating Windows–style user interface applications. A knowledgeable support network is available to supplement the extensive on–line help utility.

In Microsoft Visual C++, the compiler optimization switches are set through the "Project Settings" menu option. A specific processor architecture is selected from the following options: 80386, 80486, Pentium, Pentium Pro, and Blend. The "Blend" setting generates instructions which are fairly efficient on all processors. There are also optimization switches which fine tune program performance. Examples of these include speed maximization, code minimization, global optimizations, and improvement of floating point consistency.

With its default settings, the Microsoft Visual C++ flagged more code warnings than the DJGPP compiler. Most of these warnings were the result of data type mismatches which are said to result in inefficient code. The problems causing the warnings were corrected.

3.1.2.3 WATCOM

Rome Lab engineers requested a brief evaluation of their Watcom C/C++ 10.0 compiler. An MDEM executable was generated using this version of Watcom, yet no performance improvement was noted upon execution. The most recent version of this compiler, Watcom C/C++ 11.0, is touted for its superior code optimization capabilities. It is said to use superscalar optimization strategies to generate faster code on Pentium and Pentium Pro processors. Version 11.0 may far exceed the code optimization capabilities of its predecessor; therefore, it should not be discounted entirely.

3.1.2.4 INTEL COMPILER PLUG–IN

The Intel C/C++ compiler plug–in is used to generate machine level code which is highly optimized for a target Intel processor. It is still in the Beta phase of development and is available to users that have been accepted into the Beta program. The contractor's application for admission into the program was accepted by Intel and a Beta copy of the software was downloaded. Since the plug–in is designed to be used with the Microsoft Visual C++ Integrated Development Environment (IDE), the compiler was installed on a Pentium machine with access to the IDE at the contractor's site. The compiler has not been thoroughly tested due to a bug in the software. Intel was contacted, and a correction was suggested. Further investigation of the compiler is required in order to determine the benefits, if any, to the MDEM simulation.

3.1.3 PROFILERS

A profiler is a performance tool which assists a developer in determining which areas of code would benefit most from optimization. The profiler accomplishes this by determining the frequency of execution of selected areas of code. Typically, a profiler also maintains a record of the interrelationships between function calls. Though it is not intended to be a debugging tool, a profiler will often discover obscure bugs by exposing areas of unexecuted code which were assumed to be executed. Profilers are best used for fine–tuning large applications with many lines of code and complex data sets. In the case of MDEM, the number of execution paths is fairly manageable. It can be determined, even without the use of a profiler, that most of the execution time will be spent in the function "move_atoms", and that the optimization efforts should be focused there. However, profilers were briefly investigated to determine their specific benefits to MDEM.

3.1.3.1 DJGPP

When investigating the DJGPP profiler, it was discovered that a bug in the software made profiler operation impossible without the use of a patch. Much time was spent searching the DJGPP Mail Archives to locate the appropriate software patching instructions. Implementation of the patch required a download of the DJGPP source code and a patch utility (v2gnu/pat21b/zip). Due to time constraints, it was decided to forgo further investigation of the DJGPP profiler, especially in light of the fact that Microsoft Visual C++ tends to produce more efficient code.

3.1.3.2 MICROSOFT VISUAL C++

The profiler in Microsoft Visual C++ 5.0 contained a bug in the installation script which prevents the supporting software from acknowledging the presence of the profiler. Therefore, Version 4.2's profiling utility was analyzed instead. The instructions stated that with the proper commands, the profiler would examine only specified functions. However, the profiler was

unable to locate the user–specified starting functions, such as move_atoms. The source of this error was undetermined.

Although the "specified function" feature was not useable, the "function time coverage" option did generate the information shown in Exhibit 2. This output confirms that most of the processing time is spent in move_atoms. The functions requiring the most processing time are high–priority targets for optimization. The "Hit Count" can be used to determine which function calls should be inlined to save processing time. This listing shows force_LJ as having the highest hit count. At one point during the upgrade, the force_LJ calls were replaced by the actual force_LJ code to reduce processing time.

Exhibit 2.   Microsoft Visual C++ Profiler Output

```
Program Statistics
------------------


    Command  line  at  1997  Mar  18  13:48:  "C:\MSDEV\Projects\Er9-working\mdprof\Debug\mdprof"
9758.atm
      Total time: 121015.605 millisecond
      Time outside of functions: 15.803 millisecond
      Call depth: 3
      Total functions: 30
      Total hits: 195437
      Function coverage: 63.3%
      Overhead Calculated 8
      Overhead Average 8

Module Statistics for mdprof.exe
--------------------------------
      Time in module: 120999.801 millisecond
      Percent of time in module: 100.0%
      Functions in module: 30
      Hits in module: 195437
      Module function coverage: 63.3%

      Func              Func+Child         Hit
      Time     %          Time      %      Count  Function
   ----------------------------------------------------------------
      65548.840  54.2  107372.319   88.7      20 _move_atoms (mdemut1.obj)
      41408.035  34.2   41408.035   34.2  195160 _force_LJ (mdemut1.obj)
       7071.999   5.8    7075.674    5.8       1 _load_atoms (mdemio.obj)
       6383.855   5.3    6385.610    5.3       1 _write_data (mdemio.obj)
        317.200   0.3     317.200    0.3      20 _ul3tensor (mdemmem.obj)
        102.108   0.1     102.108    0.1       3 _atom_vector (mdemmem.obj)
         89.298   0.1      89.298    0.1      20 _free_ul3tensor (mdemmem.obj)
         30.161   0.0  120999.801  100.0       1 _main (mdem.obj)
         18.018   0.0      22.291    0.0       1 _get_header_data (mdemio.obj)
         10.088   0.0      10.088    0.0       1 _usi_vector (mdemmem.obj)
          7.948   0.0       7.948    0.0     141 _fgetline (mdemio.obj)
          5.133   0.0       5.133    0.0      20 _uli_vector (mdemmem.obj)
          2.768   0.0       2.768    0.0      20 _free_uli_vector (mdemmem.obj)
          1.103   0.0       1.103    0.0      21 _coord_vector (mdemmem.obj)
          0.917   0.0       0.917    0.0       2 _get_date_time_string (mdemio.obj)
          0.908   0.0       0.908    0.0       1 _get_yr_string (mdemio.obj)
          0.898   0.0       0.898    0.0       2 _get_time_string (mdemio.obj)
          0.417   0.0       0.417    0.0       1 _get_sec_string (mdemio.obj)
          0.106   0.0       0.106    0.0       1 _dbug (mdemut1.obj)
```

3.1.4 INTEL VISUAL TUNING ENVIRONMENT

The Intel visual tuning environment tool, VTune, is used to assist in the generation of highly optimized code targeted for Intel processor architectures. VTune has two basic modes of operation: dynamic and static. The dynamic mode is similar to a debugging or profiling environment. Capabilities include monitoring the performance of all active software, identifying "hot spots," and viewing instruction execution at the machine level. A significant learning curve is associated with the dynamic operating mode. However, the basic features of the static mode are relatively easy to use. The static mode tabulates information on code features and generates coding advice using a feature called the "Code Coach."

Since VTune is available free of charge for a 30–day trial basis, a copy was downloaded from the Internet and installed on a Pentium machine at the contractor's site. Even though the tool was not thoroughly exercised due to time constraints, it was still able to provide useful code optimization information. Valuable documentation which describes code optimization techniques was printed. In addition, the Code Coach was used to generate optimization advice for the MDEM modules with the Pentium Pro selected as the target processor.

The output of the Code Coach is presented in a window from which it is impossible to obtain a printout. This makes it difficult to scan and evaluate the advice "bullets" as a group. Much of the advice was straightforward, yet some of it was difficult to decipher. As is the case with many automated, "intelligent" tools, the user must still use discretion in following the advice. In general, references to MMX technology appeared frequently. MMX is Intel's latest technology breakthrough which is primarily intended to improve the performance of multimedia applications. The use of MMX intrinsics was recommended for array operations by the Code Coach. In addition to this, more practical, easily–implemented advice was offered. Some examples are:

- ❏ Elimination of unnecessary data type conversion

- ❏ Use of the "memset" function to initialize a block of data

- ❏ Repositioning data to avoid redundant calculations

- ❏ Changing conditional expressions to reduce the number of conditional branches

Some of this advice was tested in the move_atoms function, which resulted in a slight speedup for a large data set.

3.2 SUMMARY OF RESULTS

The degree of processing speedup attained with these code optimizations varied greatly. The most substantial improvement was seen when appropriate compiler optimization flags were used. The implementation of efficient coding practices did not result in major performance improvement, since the code was fairly clean in this area. Also, the advanced architecture of the Pentium Pro reduces some of these optimizations to insignificance. The compiler used to create an executable does have an impact on software performance. However, it was determined that the performance of the compiler optimization flags was dependent upon the data set and the changes made to the code. The executable generated with Microsoft Visual C++ was faster than the DJGPP executable for almost every test case. Though the Microsoft Visual C++ version is fast, the potential benefit of the evolving Intel compiler plug–in should not be overlooked. The profiling tools associated with DJGPP and Microsoft Visual C++ either did not work properly or did not contribute significant information. However, Intel's visual tuning environment, VTune, does generate useful optimization information.

Optimizing an algorithm for speed generally requires revision of the basic, underlying concepts or restructuring of the algorithm implementation framework. In the case of MDEM, reworking the foundation might involve implementing different methods of force calculation or integration. Thorough knowledge of supporting physics concepts is necessary to evaluate trade–offs. An alternative, faster algorithm may produce slightly different results, yet these results may fall within acceptable error limits. The other option for algorithm speedup is reassessment of the framework which implements the physical concepts. For the MDEM simulation upgrade, the software "mechanism" which performs atom pairing is the primary target for optimization. Atom pairing is responsible for the bulk of the processing load.

4.1 METHODS EMPLOYED

Two approaches were utilized to investigate algorithm optimization and enhancement. One involved a search for commercial, "stand alone" software applications which duplicated the functionality of the target algorithm, but with greater efficiency. The other area of investigation focused on incremental improvement of the algorithm framework to achieve the desired speedup.

4.1.1 COMMERCIAL SOFTWARE PACKAGES

An Internet search was performed to determine the availability of alternative, "off–the–shelf" Molecular Dynamics simulations. In general, the information gleaned from this search varied greatly in content. In some cases, the software simulations addressed technical areas outside the scope of MDEM, such as biomedical applications. In other cases, the academic community was the target audience for simulations which serve as teaching aids. There were, however, some packages which appeared to perform processing similar to that of MDEM, but with some drawbacks.

In most of the applicable commercial software, the inherent complexity seemed to extend the learning curve significantly. Additional processing features, which are not required for MDEM scenarios, constitute extra "baggage" which tend to make these simulations too cumbersome for streamlined use. Also, it may be difficult to modify a package to include custom features, since source code is not typically provided. When evaluating a commercial package, one must consider the reputation of the developer. Some software, such as "public domain," is not guaranteed to produce correct results.

Although there are initial drawbacks, the commercial packages should not be discounted completely. Given enough time to gain experience with a particular package, a user may find that it mirrors MDEM functionality while extending processing capability and offering improved performance. Table 1 presents a summary of applicable Molecular Dynamics software packages that were found during the Internet search.

Table 1. Molecular Dynamics Software Packages

Tool	Vendor	Description	Cost
HyperChem 5[3]	Hypercube, Inc.	Molecular mechanics / Dynamics modeling package	$595
HyperChem Lite[4]	Hypercube, Inc.	Subset of HyperChem 5 capabilities	$149
Simulation Kit[5]	M. Karolewski, Nanyang Tech	Programs for atomic collision simulations in solids – public domain	–

4.1.2 INCREMENTAL ALGORITHM ENHANCEMENT

The primary source of information for improvement of the MDEM algorithm was The Art of Molecular Dynamics Simulation [6] by D.C. Rapaport. This text presents information in a clear and organized format which helps the reader incorporate Rapaport's techniques into an existing simulation. C–code files which correspond directly to the text are available via Internet for use "as is." However, with the incremental improvement approach, it was more convenient to retain the MDEM code foundation and simply weave in Rapaport's recommendations where necessary, tailoring them to fit MDEM.

Use of this text had been recommended by Rome Lab engineers during the early stages of the project, initially in the context of implementing the "three–body potential" code found in Chapter 11. Synectics obtained a copy of the text and proceeded to investigate the cell method atomic neighboring schemes found in Chapter 3. These innovative algorithms reduce the number of atomic interaction computations while still maintaining data integrity. However, as with general code optimization, the speedup obtained via algorithm optimization is highly dependent on the target processor and compiler.

---

[3] HyperChem5, http://www.hyper.com/product_desc.html
[4] HyperChem Lite, http://www.hyper.com/product_desc.html
[5] Simulation Kit, http://www.uwo.ca/ssw/archives/jul96/0002.html
[6] The Art of Molecular Dynamics Simulation, D.C. Rapaport, Cambridge University Press, 1995, ISBN 0–521–44561–2

4.1.2.1 CELL IMPLEMENTATION

The atomic interaction scheme used in the baseline version of MDEM was a major improvement over earlier schemes. However, it still involved many redundant atomic pair calculations. The cell method of calculating atomic interactions, presented in Rapaport's text, appeared to have the potential to significantly reduce simulation execution time. The cell method is described as follows. The three–dimensional space spanned by the atoms is divided into *cells*, or cubes. Each cell has a side dimension which is equal to the *atomic cutoff length*. The cutoff length defines the maximum distance over which two atoms will interact. For each cell, a list is generated containing the ID numbers of all atoms that fall into the area covered by the cell. These lists are traversed when pairing atoms for the purpose of calculating the force between them.

Each cell is examined in sequence as the entire atom space is evaluated. First, all the atoms within a cell are paired with one another, avoiding redundancy. Secondly, a cell is paired with half of its neighboring cells. For each cell pairing, every atom in the first cell is paired with every atom in the second cell. If the distance between the two atoms in a pair is less than the cutoff distance, then the force is calculated and stored. Each time the force on an atom is calculated, the equal and opposite force is stored for the other atom in the pair, thus further eliminating unnecessary calculations. The cumulative forces on each atom potentially result in atomic motion. To account for atomic motion beyond cell boundaries, the cell lists must be rebuilt after each integration step.

The sample code which appears in Rapaport was tailored to suit the requirements of MDEM. Rapaport makes provision for handling periodic boundaries between cells. This is not a requirement for the current MDEM simulation. Also, Rapaport assumes that there are at least three cells in each dimension. This caused problems when testing small data sets which covered less than three cells. Code which handles less than three cells was added to the MDEM implementation of the cell method.

4.1.2.2 NEIGHBOR LIST METHOD

There are variations of the cell method which might further reduce the processing time needed to perform atomic interaction calculations. The neighbor list method, described in Rapaport's text, is one such variation. This technique assumes that the atomic motion is small enough so as not to incur major changes in the cell list even after several integration steps. The cell size would be slightly increased, by Δr, in order to provide adequate coverage of pairs which may interact after several integration steps. The method used for determining atom pairs is similar to the cell method.

In the cell method, atoms are paired and forces are immediately calculated if the distance between them is less than the cutoff distance. In the neighbor list method, atoms are paired and placed in a table (or list) if the separation between them is less than the cutoff distance plus Δr. After all potential pairs have been determined, the entire list is traversed and force calculations are made if the atoms are within interaction distance. With this method, it is not necessary to

update the neighbor list after every integration step. To determine when to refresh the neighbor list, the maximum atom displacement during each integration step is accumulated. When the accumulated value exceeds Δr, the list is recreated.

The contractor briefly experimented with this "infrequent refresh" concept using the cell method described in the previous section. With the 9,758 atom data set, the refresh time was negligible compared to the time needed to process one integration step. It was also noted that the simulation could proceed for at least 20 integration steps without refreshing the list, while still yielding valid results. Rome Lab engineers implemented the full neighbor list scheme, but had to impose certain constraints. The neighbor list method utilizes very large amounts of memory to maintain the listing of potential atom pairs. In order to execute the simulation without running out of memory, the cutoff distance was reduced, thus decreasing the potential number of valid pairs. On computers with more memory, the neighbor list method may provide significant speedup without concessions. However, given the configuration of the current platforms, it appears that the neighbor list scheme limits the simulation's flexibility.

4.2 SUMMARY OF RESULTS

Few applicable "off–the–shelf" molecular dynamics packages were discovered during the Internet search. This indicated that effort should be focused on improving the fundamental MDEM algorithm. Rapaport's molecular dynamics simulation text provided the information necessary to rework the calculation–intensive interatomic force component of the MDEM software. The baseline method of pairing atoms was replaced with the cell method, which, in addition to substantially reducing processing time, is also conducive to parallel processing implementation.

The baseline version of MDEM was a console application which executed via a command–line user interface. If errors in the atom file were flagged, the user was forced to exit the application and edit the atom file appropriately. This implementation is sufficient for experienced users, such as the Rome Lab engineers, who use the simulation on a daily basis. However, it may appear daunting to the uninitiated user. One of the goals of the MDEM simulation upgrade was to implement a prototype GUI which would provide a more user–friendly and up–to–date environment for the novice.

5.1 METHODS EMPLOYED

An Internet search revealed numerous GUI builders which are available commercially. Developers of these tools range from large, well–known corporations such as Microsoft, to smaller, emerging companies such as XVT Software. The academic community has also spawned its own versions of GUI builders, often available as public domain software. The quality of the development tools also varies dramatically, from simple and unrefined, to elaborate and highly polished.

There are independent resources, accessible via Internet, which maintain data on the latest GUI builders. One such resource, "User Interface Software Tools"[7], is a list of current GUI builders that is compiled and maintained by Brad A. Myers at Carnegie Mellon University. This list provides a very brief summary of over 100 different interface development tools. Another resource, "Graphical User Interface Development Systems"[8], is the result of research conducted by S. Baum at Texas A&M University. Though geared more toward Unix platforms, this listing of GUI builders provides useful general information as well. These summaries can serve as the starting point for future in–depth GUI builder investigations.

5.1.1 USER INTERFACE OPTIONS

The direction of the literature search for GUI builder information was driven by the needs of the MDEM simulation. Currently, MDEM has a small user–base which primarily depends on Intel

---

[7] "User Interface Software Tools", Brad A. Myers, Carnegie Mellon University,
http://www.cs.cmu.edu/afs/cs.cmu.edu/user/bam/www/toolnames.html
[8] "Graphical User Interface Development Systems", S. Baum, Texas A&M University, http://www–ocean.tamu.edu/~baum/graphics–GUI.html#wxWindows

Pentium machines to host the application. An ideal scenario would involve MDEM accessibility on a wide variety of platforms in order to accommodate a large users' group. The current and ideal situations differ drastically, thus resulting in very different requirements regarding user interface development. Given the current user situation, a platform–specific GUI builder is sufficient. Considering future goals, a cross–platform GUI builder is necessary.

Due to time constraints of the project, it was determined that the best approach would be to focus on the current user scenario by using a platform–specific GUI builder for the PC, running Windows 95/NT. In this report, information regarding cross–platform builders is also presented for future reference.

5.1.1.1 PLATFORM–SPECIFIC

Searching the Internet yielded information on several PC/Windows–based GUIL builder tools. There appeared to be more GUI builders available for Unix platforms than for PC platforms. One possible explanation might be that the Windows platform is a more recent development. Table 2 lists just a few of the packages available for use with Windows.

Table 2.   PC/Windows Platform–specific GUI Development Tools

TOOL	VENDOR	DESCRIPTION	COST
Visual C++	Microsoft	C/C++ programming environment with AppWizard Developer Tool	$479
Visual Basic	Microsoft	Typically used for prototyping interfaces	$469
GX Series Developer's Pak	Genus	Programmer's Toolkit for GUI development	$589
Rapid Design	Emultek Inc.	MS Windows, Windows 95 and NT visual language	$6,000

Microsoft Visual C++ was the primary candidate for GUI implementation since it was also the programming tool used by the contractor to implement changes in the command–line version of MDEM. A significant learning curve is associated with any GUI builder, but this was kept to a minimum due to contractor familiarity with the Microsoft Visual C++ programming environment. In addition, GUI building costs were minimized by utilizing a readily available tool. Microsoft Visual C++, in conjunction with its resident "AppWizard," helps guide the developer by providing features such as the following.

❑ Initial setup of main dialog

❑ Automatic creation of interface "shell" files

❑ File editing guidance

❑ Structures that are compatible with "Windows" format

For a realistic assessment of other GUI builders, it would be advantageous for the developer to obtain a trial copy of the software before purchase, if possible. Textual descriptions of software packages often overestimate actual software quality.

5.1.1.2 CROSS–PLATFORM TRADITIONAL

Software tools for cross–platform GUI development are continually evolving to encompass a wider range of platforms and enhance performance. Table 3 presents a sampling of the cross–platform GUI development tools found as a result of the Internet search. The tools presented here support traditional user interface development.

Table 3.   Cross–platform GUI Development Tools

TOOL	VENDOR	DESCRIPTION	COST
UIM/X with Cross–Platform Toolset	Visual Edge Software Ltd.	Cross–Platform Toolset is an add–on to UIM/X	$7,500
XVT Development Solution for C++	XVT Software	C++ GUI development tool	?
X–Designer 4	Imperial Software Technology	Used with Motif and Windows applications	$3,500
Galaxy	Visix Software Inc.	Virtual Toolkit for Mac, Windows, Motif, OpenLook	$9,600
PowerBuilder	Powersoft	PC, Unix, and Mac Interface Builder and Database front end	$2,759
XFaceMaker/Win	Nova Software Labs	Builds native GUIs with consistent appearance and behavior among Unix, Windows and Macintosh– development platform is Unix/X/Motif	?

Developers have a variety of approaches to cross–platform development.

❑ *Binary Emulation* emulates the native processor instruction set of the system for which the application was developed, allowing a program to be run on different

platforms without being recompiled. However, major drawbacks exist. Using binary emulation, causes the software to run at significantly slower speeds than native applications and requires large amounts of disk space and memory.

❑ *Layered Toolkits* implement the application in the vendor's API, which then makes calls to the underlying API of the target platform. The main advantage to the layered toolkit is that the application has the look and feel of a native application. The main disadvantage, is that the toolkit only uses the lowest common denominator features of all supported platforms. A variation of the layered toolkit is the emulation toolkit, which differs from a layered toolkit by emulating the look and feel of the native environment. The main disadvantage is that these packages are not built on the native API, requiring the vendor to reimplement the native toolkit for each target platform.

❑ *Application Frameworks* implement a set of C++ classes, much like a layered toolkit. Besides the lowest common denominator limits, application frameworks are restricted to developing the application in C++.

❑ *Ported APIs* build on the target platform's native API to provide the full range of functionality of the target platform. Ported APIs are considered among the best choice for cross–platform development.

In addition to selecting the approach and, therefore, features and functionality for the ported application, there are several other issues to be considered when designing code for cross–platform implementation. Some of the more significant issues include:

❑ *Processor Architecture* — Some processors are limited in the amount of physical and virtual memory that they can address. This becomes particularly important when large volumes of data are to be processed. Word size and byte ordering ("big–endian" as opposed to "little–endian") are also important issues to consider when data manipulation is being performed at the bit and byte level.

❑ *Operating System* — The use of threaded processing, multitasking, memory segment addressing, available services, and data protection are all impacted by the operating system to which a given application may be ported.

❑ *Memory Model* — The segmented memory model, utilized in many older (and some newer) architectures limits the maximum size of the data structures that can be utilized in an application. If portability to an architecture with a segmented memory model is required, data structures and data processing techniques must be addressed.

Furthermore, the use of vendor specific extensions, such as the Microsoft Foundation Classes extension to C++, to any programming language effectively reduce the portability of the code. Portable code can be developed using ANSI C/C++ and careful programming techniques. However, this may lead to memory management issues with the target platform. Applications

developed using inefficient compilers, such as the freeware DJGPP, running on hardware insufficient for the task can create the appearance of non–portability.

Some of these issues are addressed in several articles accessible via the Internet.[9, 10, 11, 12]

5.1.1.3 CROSS–PLATFORM BROWSER–BASED

An alternative cross–platform option is a browser–based interface, which can be crafted in a variety of ways. The least expensive way is through use of a text editor, which creates HTML page(s). With a web authoring tool, many of which are available commercially or in the public domain, a novice can quickly create a basic interface. Combining HTML with CGI programming would produce an interface which duplicates or exceeds the capabilities of the prototype delivered under this contract.

The advantages of a browser interface are:

❑ Platform independence.

❑ Smaller code requirements compared to a traditional cross–platform package.

❑ Networked processing capability.

❑ Common applications on workstations and PCs.

❑ The ability for each PC or workstation to run its own server or, the designation of a single machine to be the network's web server.

The disadvantages to a browser interface approach are few, but important, and mainly relate to the need for a web server.

❑ For the web browser to do anything other than display simple HTML pages, the browser requires the presence of a web server.

❑ The server introduces one more program to be installed, configured, and maintained.

❑ As with any networked application, multiple versions of MDEM running off a single server will slow down the execution of the application.

---

[9] "Windows Without Walls", Denis Haskin, Byte, February 1996, Special Report,
http://www.byte.com/art/9602/sec11/art3.html
[10] "Writing Portable C++ Code", Bristol Technology, Inc., http://www.bristol.com/Bibliography/xplat2.html
[11] "Serialization and MFC: Extending MFC for cross–platform portability", Bristol Technology's reprint of an article from Dr. Dobb's Journal, #229, April 1995, http://www.bristol.com/Bibiography/drdobbs0495.html
[12] "Cross Platform Toolset, Bottom Up or Top Down?", Visual Edge Software Ltd.,
http://www.vedge.com/prods/cptapp.html#anchor1594767

- With a single server, all instances of MDEM are run on the server platform. The data file may or may not be available for use. The hardware may need to be upgraded for the additional processing requirements.

- A dedicated server for each machine would require recompilation of the MDEM application for each target platform.

Despite the disadvantages, many computing locations already have a web server in place. Inexpensive web servers are available. For example, a package of WebSite 1.1 server, including user's manual, was widely available for $60. With such affordable pricing, a research environment can host web servers on each PC. WebSite, for example, would require little in the way of hardware upgrades to most moderately equipped PCs.

5.1.1.3.1 Java

Given the browser–based approach to a GUI, Java presents itself as one possible solution. Java is a simple, object–oriented, platform–independent programming language specifically geared towards creating small programs, or applets. It can be a standalone program or work in conjunction with the web browser.

Using Java to produce a GUI to MDEM has several advantages.

- Java is a bare–bones version of C++. The learning curve, therefore, is similar to that of learning to use the C++ libraries in Microsoft Visual C++.

- The Java source code is interpreted into byte code which the Java Virtual Machine (JVM) runs. The JVM is an integrated part of the web browser. Therefore, Java applets are written once, using whatever platform is desired. The applet can then be used on any Java supporting browser.

- The Java applet code does not interact directly with the MDEM code, leaving MDEM available for changes.

- With Java development tools the browser interface can be quickly prototyped and implemented.

The disadvantages of a Java interface to MDEM are closely linked to the advantages.

- Java is another programming dialect. It introduces another layer of maintenance.

- Currently, the Air Force is hesitant to accept Java due to security concerns. As MDEM is a research tool and not a deployable system, security should not be an issue.

- What Java gains in portability over C++, it loses in speed.

5.1.1.3.2   Java Tools

Java tools come in a variety of flavors. Essentially, these tools can be grouped into five categories.

- ❏ Interactive Development Environment (IDE)

- ❏ Authoring and Animation Tools

- ❏ Development Tools

- ❏ Database Connectivity Tools

- ❏ Miscellaneous

Java tools also provide "wizards" which let the programmer decide how little or how much of the code to actually write. Table 4 represents a small sampling of the various Java tools[13] .

Table 4.   Java Developer Tools

PRODUCT	VENDOR	DESCRIPTION	COST
Visual Café Pro	Passport Corporation	IDE that includes visual programming tools	$8,995
Powerblend	CreativeSoft	Package provides reusable, extensible components that supply tool bars, status meters, message boxes, combo boxes, etc.	$250
Visual J++ Professional	Microsoft	GUI language IDE for writing and integrating Java	$99
Café	Symantec	Builds entire working code or just the outline through the use of wizards	$80

5.1.2 MDEM PROTOTYPE USER INTERFACE IMPLEMENTATION

A prototype version of the MDEM user interface was created using the Microsoft Visual C++ AppWizard. The purpose of the prototype was to demonstrate basic functionality and to generate discussion for future enhancements. Throughout the upgrade, feasibility analyses were

---

[13] Annual Java Issue. WebDeveloper, May/June 1997 issue, pp. 38–45

conducted to determine the practicality and usefulness of proposed interface features. After discussions with Rome Lab, it was concluded that the user interface would exhibit only basic functionality, but not advanced features, given the time constraints. The prototype user interface serves as a model which could be enhanced at a later date when the end–user group expands. Exhibit 3 provides a view of the initial screen of the MDEM GUI prototype.

The GUI version of MDEM contains all of the functionality of the command–line version, in addition to features that allow the user to interactively modify input data. The values of all fields on the interface, except execution time, system initial temperature, and pinning file name fields, can be changed directly on the interface. This provides the flexibility of running simulations with minor to major changes in the system parameters, without the need to open the input file to change header data. The capabilities of the GUI version are as follows.

❑ *Main Functions*

◆ Load Data — Loads the data from any valid *.atm file. Uses the Windows' standard Open dialog, allowing the user to traverse the machine's directory.

◆ Save Data — Saves to a new file the inputs seen on the interface, along with the original atom data loaded. Uses the Window's standard Save As dialog.

◆ Reset Data — Restores the original values for the current data set. Once the data has been saved as a new input file, the new input values become the "original".

◆ Run Mdem — Processes the integration. If any changes to the original values are detected, the Save As dialog automatically displays.

◆ Cancel Mdem — Cancels the current operation. If the integration is in progress, Mdem finishes the current integration step and writes out the system data to the output file.

◆ Exit Mdem — Exits the program.

◆ Help — Displays minimal information on the application.

❑ *View Atoms* — Displays the number of atoms by type comprising the data set.

❑ *View Thermostat Data 1–7* — Displays the values for the seven initial thermostats in a pop–up window including xi0, temp_ref, temp_step, total work, and temp_stat.

❑ *Pinning*

◆ Change Pinning File — Displays the name of the pinning file associated with the loaded data set, if any. The user can select this button to display the Open dialog, for selecting or changing a pinning file name.

◆ View Pinning Points — Displays the x, y, and z components of the pinning points in a pop–up window.

◆ <u>View</u> <u>Pinning</u> <u>Constants</u> — Displays the x, y, and z components of the pinning force constants in a pop–up window.

❑ *Status Area*

◆ <u>Processing</u> <u>Messages</u> — Displays processing messages such as detection of invalid data during the load process, current integration step number, and mean compute time per sec of simulation time for a system of "n" atoms.

◆ <u>Processing</u> <u>Meter</u> — Provides visual confirmation that the system is still working.

5.2 SUMMARY OF RESULTS

The GUI created for the MDEM software was intended to be a prototype which would foster discussion and definition of future user interface requirements. The prototype was developed using Microsoft Visual C++ due to time and budget constraints. However, this is not necessarily the best tool for future development, given the wide variety of cross–platform GUI–building tools commercially available.

Exhibit 3.   MDEM GUI Prototype

Requirements for the MDEM simulation upgrade evolved as the task progressed. After significant speedup improvements had been achieved, work became increasingly focused on features which would improve the overall quality and flexibility of the simulation. Also, simulation accuracy was scrutinized more closely by observing the results obtained with very small data sets. Each interim version of the MDEM upgrade software sparked new discussions and requirements analysis.

6.1 METHODS EMPLOYED

The additional enhancements incorporated into the MDEM software ranged from major changes, which affected many source code functions, to relatively minor, localized modifications. These enhancements included:

- ❑ Atom pinning
- ❑ Double precision conversion
- ❑ Atom count message
- ❑ Drag–and–drop input
- ❑ Optional temperature file output
- ❑ User interrupt processing

The importance of these additions also varied significantly. For example, double precision conversion was essential for accurate simulation processing, while drag–and–drop was implemented primarily for convenience.

6.1.1 ATOMIC PINNING FEATURE

The concept of pinning can be described as "tethering" an atom, as if by a spring, to a point, line, or plane in three–dimensional space. The force constant associated with the "spring" is called the pinning force constant, and it is in the direction of the point, line, or plane. For a pinning point, the spring connects the atom and the point. For pinning to a line or plane, one end of the spring is on the atom, and the spring is orthogonal to the line or plane. Pinning to a line or plane would eliminate the need to calculate individual pinning points and would simulate a force which could affect many atoms in the sample space. However, individual pinning points provide the user with greater overall flexibility.

The baseline version of MDEM contained references to pinning data, but did not contain operational pinning code. In this version, a total of seven pinning planes and seven pinning constants were stored in array format. For pinned atoms, this array was indexed by the thermostat value. Pinned atoms were identified by an encoded "1" in the atoms tag data. During the upgrade effort, Rome Lab engineers decided to have pinning incorporated into the software. The requirements specified replacement of the seven pinning planes with potential pinning points for each atom. Also, the single pinning force constant was replaced with pinning force x, y, and z components. The array format of the pinning constants, indexed by thermostat value, was retained. These features offer more flexibility to the user, though data creation is made more complicated.

The contractor implemented the changes needed to incorporate pinning. A keyword was added to the atom file header to indicate the presence of a pinning point file. If the keyword is present, the text string following it is the name of the pinning point file. The file is assumed to reside in the same directory as the atom file. The pinning point file contains one line of data for each atom that is pinned. Each line contains the atom sequential identification number, derived from the atom file, and the x, y, and z coordinates of the pinning point for that atom. Basic error checking is performed when the software reads in the data sets.

The data in the pinning point file is read into an array which is dimensioned for the entire set of atoms. Therefore, if only a subset of the atoms is pinned, this array becomes a sparse array. This method of pinning point implementation may require more memory than is necessary, but is intended to minimize processing time. An alternative implementation would require usage of a pointer to the next pinned atom to be processed, thus eliminating sparse array usage. A comparison of the execution times for each method and an assessment of the tradeoffs would be necessary in selecting the most appropriate method. A preliminary implementation using the sparse array was given to Rome Lab as a working copy for evaluation; it was deemed acceptable for this effort.

6.1.2 DOUBLE PRECISION CALCULATIONS

During the upgrade effort, Rome Lab engineers noticed that MDEM was erroneously generating zero outputs for the 13–atom data set. To assist in finding the problem, the contractor validated the neighbor list for each atom, ensuring that all atom pairs were being considered in this small set. The atomic pairing mechanism was found to be working properly. Rome Lab suggested that the single precision floating point implementation prevalent throughout the simulation might be the cause of the problem, since it would obscure very small atomic motion deltas. Therefore, the contractor converted all single precision floating point variables to double precision format.

The change to double precision format corrected the problem of undetectable motion deltas. However, the execution time of the double precision version of MDEM is significantly longer than that of the single precision version. Also, when compiled under DJGPP, the resulting

executable requires greatly increased amounts of memory to execute. Despite the processing time and memory setbacks, the necessity of the conversion is evidenced in the plots of temperature data. Exhibit 4 shows a plot of temperature vs. integration step for the single precision floating point version of MDEM. There are many linear segments on the curve, indicating atomic motion so minuscule that it was undetected. Exhibit 5 is a plot of temperature vs. integration step for the double precision version. The resulting curve is smooth, more accurately reflecting changes in the atomic motion.

Exhibit 4. Output of Single Precision Version of MDEM
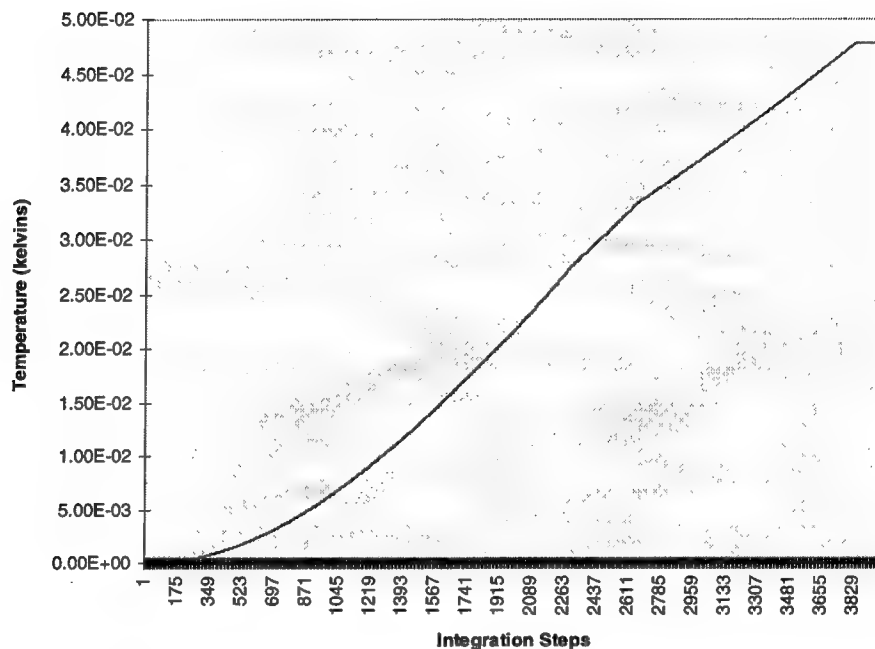
6.1.3 ATOM TYPE MESSAGE

During execution of the command–line version of MDEM, the atom header data are echoed to the console for the user for verification. The atom data (position, velocity, and tag data), due to its potential size, is not written to the console. Rome Lab engineers indicated that it would be useful to have a summary of the atom data displayed to the user. The contractor implemented a feature which displays a message indicating the types of atoms included in the current data set. This information is derived from the atomic number which is extracted from the encoded atom tag. The number of each type of atom is also displayed. This atom type message allows the user to verify, at a high level, that the correct data set has been selected.

Exhibit 5.   Output of Double Precision Version of MDEM
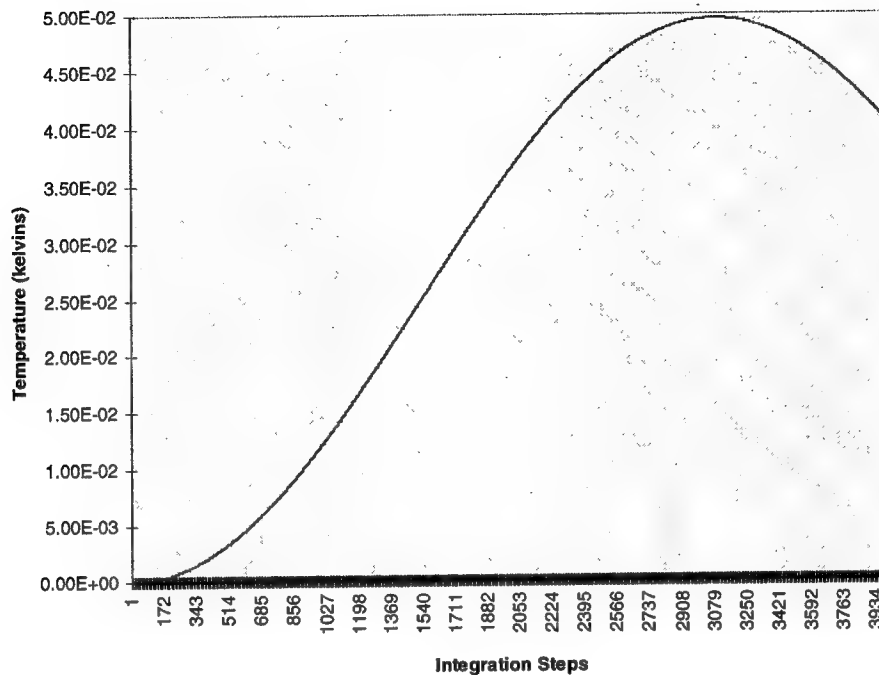
6.1.4 DRAG–AND–DROP FEATURE

Rome Lab engineers had suggested that a Windows–style "drag–and–drop" capability would be useful for executing the command–line version of MDEM. The drag–and–drop feature would allow the user to drag an atom file icon and drop it on the MDEM executable. This action would then start program execution. The contractor implemented a drag–and–drop feature, thus allowing the user to bypass the typing associated with the command–line interface. However, this implementation does not provide a means to write messages to screen during execution. Instead, output is automatically placed in a "Windows" directory.

6.1.5 OPTIONAL OUTPUT OF TEMPERATURE FILE

With the baseline version of the MDEM code, the temperature file, or *.tm file, was generated every time the simulation was executed. At Rome Lab's request, output of the temperature file was made optional.

6.1.6 USER INTERRUPT PROCESSING

A user–interrupt handling routine was included in the baseline version of MDEM, but it was not functional. Rome Lab suggested that a user interrupt handler be implemented that would allow processing to continue through the current integration step and then save the atom data to a file. The contractor incorporated this user–interrupt processing in both the command–line and GUI versions of MDEM. This feature allows the user to cancel a lengthy simulation without losing the state variables of the system. Simulation processing can then resume with the next integration step at a later time.

6.2 SUMMARY OF RESULTS

The additional software enhancements presented in this section were incorporated to either increase MDEM functional capability, provide accurate results, or improve user interaction. Their implementation contributed to the overall robustness and user–friendliness of the MDEM application.

Benchmarking occurred after each iteration of MDEM modification, or "build." In addition to tracking changes to the code and algorithm, execution times resulting from the various compiler optimization flags were also recorded. Table 5 shows the evolution of the MDEM software with respect to the incremental builds that were delivered to Rome Lab. The changes made in Mdem2 degraded performance and were not included in subsequent builds.

Table 5.   MDEM Build History

BUILD	CHANGE DESCRIPTION
Baseline	Original code provided by Rome Lab
Mdem0	Corrected data type mismatches to prevent potential inaccuracies
Mdem1	Basic coding optimizations as a result of visual inspection
Mdem2	Primitive data input as preliminary block I/O implementation, one–time atom tag decode
Mdem3	Cell method of interatomic force calculation, pinning
Mdem4	Conversion to double precision, drag–and–drop, atom count message, user–selectable temperature file
Mdem_GUI	Graphical user interface

7.1 METHODS EMPLOYED

Benchmarking results were recorded for two data sets. The first one consists of 9,758 Nickel atoms. The second data set is a box about 70 x 106 x 140 angstroms, filled with 112,423 FCC Nickel atoms. Twenty integration steps were executed for both data sets. Table 6 summarizes the results obtained on a Pentium Pro machine (200 MZ with 64 MB RAM, running Windows NT 4.00.1381). Table 7 summarizes the results obtained on a Pentium machine (75 MZ with 24 MB RAM, running Windows 95). Both the DJGPP and Microsoft Visual C++ executables were used for benchmarking. Some benchmarks are not available for the Pentium processor, due to memory limitations of the computer or time constraints of the upgrade effort.

Table 6.　　Pentium Pro Benchmarks

Compiler	Version	Optimization	9,758 Atoms		112,423 Atoms	
			Total (sec)	integration time-step = 2 fs	Total (sec)	integration time-step = 1 fs
DJGPP						
	baseline	unknown	52	2.6000e+00	605	3.025e+01
	Mdem0					
		−O1	38	1.9000e+00	451	2.255e+01
		−O2	42	2.100e+00	496	2.480e+01
	Mdem1					
		−O1	48	2.400e+00	572	2.860e+01
		−O2	40	2.000e+00	474	2.370e+01
	Mdem3					
		−O1	26	1.300e+00	358	1.790e+01
		−O2	28	1.400e+00	385	1.925e+01
	Mdem4					
		−O1	28.63	1.431e+00	396.70	1.984e+01
		−O2	20.99	1.049e+00	286.81	1.434e+01
MVC++						
	Mdem0					
		BG[14]	21	1.050e+00	252	1.260e+01
		BM[15]	21	1.050e+00	252	1.260e+01
		PrG[16]	20	1.000e+00	234	1.170e+01
		PrM[17]	20	1.000e+00	236	1.180e+01

[14] BG — Blend, Global
[15] BM — Blend, Maximize Speed
[16] PrG — Pentium Pro, Global
[17] PrM — Pentium Pro, Maximize Speed

Table 6.   Pentium Pro Benchmarks (cont'd)

Compiler	Version	Optimization	9,758 Atoms		112,423 Atoms	
			Total (sec)	integration time-step = 2 fs	Total (sec)	integration time-step = 1 fs
MVC++ (cont'd)						
	Mdem1					
		BG	21	1.050e+00	251	1.255e+01
		BM	21	1.050e+00	250	1.250e+01
		PrG	20	1.000e+00	234	1.170e+01
		PrM	20	1.000e+00	236	1.180e+01
	Mdem3					
		BG	12	6.000d–01	165	8.250e+00
		BM	12	6.000e–01	163	8.150e+00
		PrG	12	6.000e–01	164	8.200e+00
		PrM	12	6.000e–01	160	8.000e+00
	Mdem4					
		BG	14.40	7.200e–01	191.64	9.582+00
		BM	22.26	1.113e+00	304.64	1.523e+01
		PrG	17.03	8.517e–01	228.74	1.144e+01
		PrM	21.40	1.070e+00	292.50	1.463e+01
	Mdem_GUI					
		BG	20	1.000e+00	275	1.375e+01
		BM	20	1.000e+00	268	1.340e+01
		PrG	22	1.100e+00	230	1.150e+01
		PrM	16	8.000e–01	321	1.605e+01

Table 7.   Pentium Benchmarks

Compiler	Version	Optimization	9,758 Atoms		112,423 Atoms	
			Total (sec)	integration time-step = 2 fs	Total (sec)	integration time-step = 1 fs
DJGPP						
	baseline	unknown	101	5.050e+00	1700	8.500e+01
	Mdem0					
		–O1	98	4.900e+00	1638	8.190e+01
		–O2	101	5.050e+00	1705	8.525e+01
	Mdem1					
		–O1	97	4.850e+00	1638	8.190e+01
		–O2	100	5.000e+00	1704	8.520e+01
	Mdem3					
		–O1	59	2.950e+00	1252	6.260e+01
		–O2	60	3.000e+00	1186	5.930e+01
	Mdem4					
		–O1	82.31	4.115e+00	–	insufficient memory
		–O2	80.16	4.008e+00	–	insufficient memory
MVC++						
	Mdem0					
		BG	91	4.550e+00	1034	5.170e+01
		BM	91	4.550e+00	1032	5.160e+01
		PG[18]	88	4.400e+00	1050	5.250e+01
		PM[19]	88	4.400e+00	1055	5.275e+01

---

[18] PG — Pentium, Global
[19] PM — Pentium, Maximize Speed

Table 7.   Pentium Benchmarks (cont'd)

Compiler	Version	Optimization	9,758 Atoms		112,423 Atoms	
			Total (sec)	integration time-step = 2 fs	Total (sec)	integration time-step = 1 fs
MVC++ (cont'd)						
	Mdem1					
		BG	90	4.500e+00	1030	5.150e+01
		BM	90	4.500e+00	1028	5.140e+01
		PG	92	4.600e+00	1050	5.250e+01
		PM	90	4.500e+00	1047	5.235e+01
	Mdem3					
		BG	52	2.600e+00	666	3.330e+01
		BM	52	2.600e+00	664	3.320e+01
		PG	52	2.600e+00	708	3.540e+01
		PM	52	2.600e+00	684	3.420e+01
	Mdem4					
		BG	66.07	3.304e+00	1037.82	5.189e+01
		BM	81.62	4.081e+00	1212.92	6.065e+01
		PG	62.50	3.125e+00	1042.65	5.213e+01
		PM	77.22	3.861e+00	1236.93	6.185e+01
	Mdem_GUI					
		BG	84	4.200e+00	1169	5.845e+01
		BM	84	4.200e+00	1257	6.285e+01
		PG	76	3.800e+00	1251	6.255e+01
		PM	73	3.650e+00	1220	6.100e+01

7.2 SUMMARY OF RESULTS

Speedup values were calculated from the timing information presented in the previous section. Speedup is determined by dividing the execution time of an older build of code by the execution time of the new and "improved" build. For the MDEM study, the lowest execution time, corresponding to the optimal compiler switch for a build, was used in the speedup calculation. The speedup values are interpreted as follows.

❑ Greater than 1 — performance improved

❑ Equal to 1 — performance not affected

❑ Less than 1 — performance degraded

Tables 8 through 11 present the speedup data for the MDEM simulation upgrade. The change in performance is directly related to the processor, data set, compiler, compiler switches, and nature of the changes made to the code. (The MDEM build descriptions are listed in Table 5.) Speedup for the GUI version of MDEM is not included in the following tables. The GUI version is approximately 10 to 20 percent slower than the command–line version, due primarily to program interaction with the main dialog.

Table 8.   Speedup for Pentium Pro, DJGPP Executable

ATOMS	9,857				112,423			
BUILD	0	1	3	4	0	1	3	4
baseline	1.37	1.30	2.00	2.48	1.34	1.28	1.69	2.11
0	–	0.95	1.46	1.81	–	0.95	1.26	1.57
1	–	–	1.54	1.91	–	–	1.32	1.65
3	–	–	–	1.24	–	–	–	1.25

Table 9.   Speedup for Pentium Pro, Microsoft Visual C++ Executable

ATOMS	9,857				112,423			
BUILD	0	1	3	4	0	1	3	4
base	2.60	2.60	4.33	3.61	2.59	2.59	3.78	3.16
0	–	1.00	1.67	1.39	–	1.00	1.46	1.22
1	–	–	1.67	1.39	–	–	1.46	1.22
3	–	–	–	0.83	–	–	–	0.83

Table 10. **Speedup for Pentium, DJGPP Executable**

ATOMS	9,857				112,423			
BUILD	0	1	3	4	0	1	3	4
base	1.03	1.04	1.71	1.26	1.04	1.04	1.43	N/A
0	–	1.01	1.66	1.22	–	1.00	1.38	N/A
1	–	–	1.64	1.21	–	–	1.38	N/A
3	–	–	–	0.74	–	–	–	N/A

Table 11. **Speedup for Pentium, Microsoft Visual C++ Executable**

ATOMS	9,857				112,423			
BUILD	0	1	3	4	0	1	3	4
base	1.11	1.12	1.94	1.52	1.65	1.65	2.56	1.64
0	–	1.01	1.75	1.38	–	1.00	1.55	0.99
1	–	–	1.73	1.36	–	–	1.55	0.99
3	–	–	–	0.79	–	–	–	0.64

The MDEM simulation upgrade revealed many interesting facts which were not entirely known at the start of the task. The following list presents a summary of lessons learned during the upgrade.

- ❑ Code which produces valid results with one compiler may not produce valid results with other compilers.

- ❑ The use of block I/O to handle atom data does not result in a significant speedup given the current ASCII text data format.

- ❑ Consolidation of atom tag masks and shifts does not reduce processing time.

- ❑ In MDEM, algorithm changes account for the most significant speedup, since the baseline code was already fairly clean.

- ❑ The speedup attained via code or algorithm improvements varies significantly based on target processor architecture.

- ❑ Code enhancements for speedup which would have made a significant impact on earlier processors, are almost insignificant on the processor architectures of today.

- ❑ The Microsoft Visual C++ implementation of the GUI interface should be multithreaded.

During the MDEM simulation upgrade, numerous topics were investigated and many software improvements were implemented. Due to time constraints, some very promising areas were only briefly examined. Potential areas for future research and development are summarized below.

❑ *Code Optimization*

◆ Function inlining

◆ Enhanced error checking

◆ Flexible pinning file location

◆ Thorough testing of MDEM code

◆ Data storage scheme evaluation to avoid the use of exponent and mantissa implementation inherent in double precision mode

◆ Watcom 11.0 evaluation

◆ Continued investigation of Intel compiler plug–in

◆ Microsoft Visual C++ purchase for primary development

◆ VTune purchase and regular usage

◆ Investigation of MMX technology for speedup

◆ Reclassification of data in header of *.atm file to list more appropriate data under "control" and "status"

❑ *Algorithm Optimization*

◆ Improvement of current pinning implementation

◆ Implementation of pinning to lines and planes

◆ Incorporation of additional features found in Rapaport's text

◆ Evaluation of public domain software "Simulation Kit"

◆ Investigation of parallel processing options using the Rome Lab Paragon system

◆ Investigation of Pentium dual–processor systems

❑ *User Interface Enhancement*

◆ Refinement of prototype user interface

■ improved pinning point display

- improved atom data display
- enhanced help utility
- "print file" capability
- user selection of output file and directory
- use of compiler directives to maintain one version of MDEM for command line and GUI
- multithreaded approach

◆ Implementation of cross–platform or web browser interface

◆ Creation of "end–to–end" interface which encompasses data synthesis, MDEM processing, and data visualization

❑ *Additional Software Enhancements*

◆ Implementation of deposition code which adds atoms to the initial data set as the simulation progresses

This simulation upgrade effort has established the foundation for future refinement and enhancement of MDEM software. The optimization of an evolving system, such as MDEM, is inherently iterative. As new functionality is added to the simulation, the underlying code must be inspected for optimal implementation. No hard and fast statement can be given regarding which combination of processor/optimization options produced the fastest executable. In general though, the processor option "Blend" and the optimization "Global" produced the fastest code for both the Pentium and the Pentium Pro. MDEM.exe and MDEM_GUI.exe, based on the double precision version (Build 4), were delivered using these options.

In today's rapidly changing technological environment, advances may be made which require the reassessment of improvements made during this upgrade. Periodic literature and Internet searches should be performed to stay attuned to the latest developments in applicable technology. Dedication to continual improvement will ensure the future success of the MDEM simulation.

APPENDIX A
ACRONYMS

CGI	Common Gateway Interface
GUI	Graphical User Interface
HTML	HyperText Markup Language
IDE	Integrated Development Environment
MDEM	Molecular Dynamics of Electronic Materials
MFC	Microsoft Foundation Class
MVC++	Microsoft Visual C++

MISSION
OF
ROME LABORATORY

Mission. The mission of Rome Laboratory is to advance the science and technologies of command, control, communications and intelligence and to transition them into systems to meet customer needs. To achieve this, Rome Lab:

    a. Conducts vigorous research, development and test programs in all applicable technologies;

    b. Transitions technology to current and future systems to improve operational capability, readiness, and supportability;

    c. Provides a full range of technical support to Air Force Material Command product centers and other Air Force organizations;

    d. Promotes transfer of technology to the private sector;

    e. Maintains leading edge technological expertise in the areas of surveillance, communications, command and control, intelligence, reliability science, electro-magnetic technology, photonics, signal processing, and computational science.

The thrust areas of technical competence include: Surveillance, Communications, Command and Control, Intelligence, Signal Processing, Computer Science and Technology, Electromagnetic Technology, Photonics and Reliability Sciences.